why should I make a copy of a data frame in pandas

Asked 5 years, 4 months ago Active 11 months ago Viewed 139k times

169

41

When selecting a sub dataframe from a parent dataframe, I noticed that some programmers make a copy of the data frame using the `.copy()` method.

Why are they making a copy of the data frame? What will happen if I don't make a copy?

python pandas chained-assignment

edited Mar 7 '19 at 0:29
JJ for Transparency and Monica
841 ● 6 ● 14 ● 26

asked Dec 28 '14 at 2:22
Elizabeth Susan Joseph
3,735 ● 5 ● 15 ● 21

6 My guess is they are taking extra precaution to not modify the source data frame. Probably unnecessary, but when you're throwing something together interactively, better safe than sorry. – Paul H Dec 28 '14 at 2:41

7 I assume this not a stupid question for giving a negative one. – Elizabeth Susan Joseph Dec 28 '14 at 10:38

5 Answers

Active Oldest Votes

190

This expands on Paul's answer. In Pandas, indexing a DataFrame returns a reference to the initial DataFrame. Thus, changing the subset will change the initial DataFrame. Thus, you'd want to use the copy if you want to make sure the initial DataFrame shouldn't change. Consider the following code:

```
df = DataFrame({'x': [1,2]})
df_sub = df[0:1]
df_sub.x = -1
print(df)
```

You'll get:

```
x
0 -1
1  2
```

In contrast, the following leaves df unchanged:

```
df_sub_copy = df[0:1].copy()
df_sub_copy.x = -1
```

answered Dec 28 '14 at 20:01

cgold
2,063 ● 1 ● 9 ● 7

5 is this a deep copy? – bikashg May 11 '19 at 23:05

5 Yes. The default mode is "deep" copy! pandas.pydata.org/pandas-docs/stable/reference/api/... – Ambareesh May 19 '19 at 4:26

42

Because if you don't make a copy then the indices can still be manipulated elsewhere even if you assign the dataFrame to a different name.

For example:

```
df2 = df
func1(df2)
func2(df)
```

func1 can modify df by modifying df2, so to avoid that:

```
df2 = df.copy()
func1(df2)
func2(df)
```

answered Sep 22 '16 at 1:27

sparrow
5,688 ● 7 ● 31 ● 57

Wait wait wait, can you explain WHY this occurs? Doesn't make sense. – NoName Feb 10 at 7:00

2 it is because in the first example, ` df2 = df `, both variables reference the same DataFrame instance. So any changes made to df` or `df2` will be made to the same object instance. Whereas in the `df2 = df.copy()` a second object instance is created, a copy of the first one, but now `df` and `df2` reference to different object instances and any changes will be made to their respective DataFrame instance. – Pedro Feb 13 at 3:36

15

It's necessary to mention that returning copy or view depends on kind of indexing.

The pandas documentation says:

> Returning a view versus a copy
>
> The rules about when a view on the data is returned are entirely dependent on NumPy. Whenever an array of labels or a boolean vector are involved in the indexing operation, the result will be a copy. With single label / scalar indexing and slicing, e.g. df.ix[3:6] or df.ix[:, 'A'], a view will be returned.

answered Jan 20 '17 at 13:22

Gusev Slava

1,340 ● 1 ● 13 ● 20

pandas.pydata.org/pandas-docs/version/0.12/… and newer version pandas.pydata.org/pandas-docs/stable/user_guide/… – vasili111 Apr 22 at 15:51

9

The primary purpose is to avoid chained indexing and eliminate the `SettingWithCopyWarning`.

Here chained indexing is something like `dfc['A'][0] = 111`

The document said chained indexing should be avoided in Returning a view versus a copy. Here is a slightly modified example from that document:

```
In [1]: import pandas as pd

In [2]: dfc = pd.DataFrame({'A':['aaa','bbb','ccc'],'B':[1,2,3]})

In [3]: dfc
Out[3]:
    A   B
0   aaa 1
1   bbb 2
2   ccc 3

In [4]: aColumn = dfc['A']

In [5]: aColumn[0] = 111
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

In [6]: dfc
Out[6]:
    A   B
0   111 1
1   bbb 2
2   ccc 3
```

Here the `aColumn` is a view and not a copy from the original DataFrame, so modifying `aColumn` will cause the original `dfc` be modified too. Next, if we index the row first:

```
In [7]: zero_row = dfc.loc[0]

In [8]: zero_row['A'] = 222
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

In [9]: dfc
Out[9]:
    A   B
0   111 1
1   bbb 2
2   ccc 3
```

This time `zero_row` is a copy, so the original `dfc` is not modified.

From these two examples above, we see it's ambiguous whether or not you want to change the original DataFrame. This is especially dangerous if you write something like the following:

```
In [10]: dfc.loc[0]['A'] = 333
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

In [11]: dfc
Out[11]:
    A   B
0   111 1
1   bbb 2
2   ccc 3
```

This time it didn't work at all. Here we wanted to change `dfc`, but we actually modified an intermediate value `dfc.loc[0]` that is a copy and is discarded immediately. It's very hard to predict whether the intermediate value like `dfc.loc[0]` or `dfc['A']` is a view or a copy, so it's not guaranteed whether or not original DataFrame will be updated. That's why chained indexing should be avoided, and pandas generates the `SettingWithCopyWarning` for this kind of chained indexing update.

Now is the use of `.copy()`. To eliminate the warning, make a copy to express your intention explicitly:

```
In [12]: zero_row_copy = dfc.loc[0].copy()

In [13]: zero_row_copy['A'] = 444 # This time no warning
```

Since you are modifying a copy, you know the original `dfc` will never change and you are not expecting it to change. Your expectation matches the behavior, then the `SettingWithCopyWarning` disappears.

Note, If you do want to modify the original DataFrame, the document suggests you use `loc`:

```
In [14]: dfc.loc[0,'A'] = 555

In [15]: dfc
Out[15]:
    A   B
0   555 1
1   bbb 2
2   ccc 3
```

edited May 31 '19 at 2:05

answered Oct 22 '18 at 9:58

Cosyn
2,099 ● 18 ● 18

1

In general it is safer to work on copies than on original data frames, except when you know that you won't be needing the original anymore and want to proceed with the manipulated version. Normally, you would still have some use for the original data frame to compare with the manipulated version, etc. Therefore, most people work on copies and merge at the end.

answered Mar 28 '18 at 23:31

bojax
21 ● 7